Universitatea Tehnică din Cluj-Napoca

Simularea software

a unui utilaj cu comandă numerică

Sorana Ghiorghe

30235

Proiect pentru Structura Sistemelor de Calcul

Contents

1 Introducere

1.1 Context

În cadrul acestui proiect, se urmărește dezvoltarea unei **simulări pentru un utilaj de tăiat cu flama controlat numeric (CNC)**. Scopul principal este de a reproduce mișcările și comenzile specifice unui astfel de utilaj, utilizând instrucțiuni din **fisiere G-code** pentru a genera traiectorii de tăiere pe axele **X** și **Y**. Această simulare va oferi o vizualizare în timp real a deplasării capului de tăiere și a procesului în sine, punând în evidență funcționarea unei masini CNC.

1.2 Obiective

Acest proiect are ca scop principal dezvoltarea unei astfel de simulări software, în care traiectoriile și comenzile pentru utilaj sunt prelucrate și vizualizate într-un mediu virtual.

1. Crearea unei soluții pentru citirea și interpretarea fișierelor G-code, având abilitatea să identifice instrucțiunile scrise în acest limbaj pentru diferite mișcări și comenzi.
2. Generarea comenzilor de miscare pentru utilaj astfel încat acesta să se poată deplasa corect pe axele X și Y și să respecte în intregime condițiile impuse.
3. Simularea grafică a procesului de tăiere prin crearea unei interfețe care să ofere posibilitatea vizualizării procesului de taiere in timp real.

2 Studiu Bibliografic

2.1 Utilajul cu comanda numerica

Utilajele de tăiat cu flama controlate numeric sunt esențiale în industria prelucrării metalelor, fiind utilizate pentru tăierea precisă a oțelului și a altor materiale metalice dure. Principiul de bază constă în utilizarea unei flăcări intense pentru a încălzi materialul la temperaturi apropiate de punctul de topire, urmată de injectarea unui flux de oxigen pur care arde materialul, transformându-l în oxid de fier care este apoi eliminat din zona de tăiere.

Controlul utilajelor CNC se face evident prin comenzi numerice, iar limbajul utilizat de acestea pentru a interpreta instrucțiuni este important pentru modul lor de operare. Principalul limbaj utilizat este cunoscut drept G-code, care furnizeaza informații relevante pentru utilaj.

Simularea software a funcționării acestor utilaje oferă un mediu sigur pentru testarea algoritmilor de control, vizualizarea traiectoriilor de tăiere și analiza eventualelor erori fără riscul de a deteriora materialul sau utilajul.

2.2 G-Code

G-code este un limbaj de programare standardizat utilizat pentru a controla utilajele cu comandă numerică computerizate (CNC), cum ar fi mașinile de tăiat sau imprimantele 3D. Acesta oferă instrucțiuni mașinii legate de mișcările liniare și circulare de realizat, viteza de deplaseze și ce acțiuni să efectueze pe parcursul procesului de lucru.

Denumirea de G-code provine din natura sa geometrică. Instrucțiunile din G-code indică mașinii cum să se deplaseze în sistemul de coordonate carteziene, însă pe lângă acestea, G-code furnizează și alte

informații esențiale: viteza de deplasare, unghiul de rotație, offsetul lungimii, punctele de pornire și oprire, *feed rate*, timpul de așteptare etc. Acesta este utilizat în combinație cu M-Code, care controlează funcțiile operaționale ale utilajului, precum sfârșitul programului sau alte acțiuni.

Alfabetul comun pentru G-code:

- G: Mișcări generale ale mașinii
- F: Feed rate
- T: Schimbarea uneltei
- S: Spindle speed
- X, Y, Z: Axele liniare în sistemul cartezian de coordonate.
- A, B, C: Axele de rotatie în jurul X, Y, Z

Un exemplu de cod care utilizează G-code:

```
G21; Setează dimensiunile în milimetri
G90; Setează sistemul la o poziționare absolută
G00 Z5; Ridică unealta la o înălțime de 5 mm deasupra piesei de prelucrat
G00 X0 Y0 ; Poziționarea rapidă a uneltei în punctul origine
G01 Z-1 F100; Coboară unealta la o adâncime de 1mm cu o viteză de avans de 100mm/min
G01 X20 F200 ; Mută unealta la coordonata X= 20 cu o viteză de avans de 200mm/minut
G01 Y20 ; Mută unealta la coordonata Y= 20
G01 X0 ; Mută unealta la coordonata X= 0
G01 Y0 ; Mută unealta la coordonata Y= 0
G00 Z5; Ridică unealta la înălțimea de siguranță de 5 mm
M0; Oprirea programului
```

3 Analiza

3.1 Propunere proiect

Simularea va include următoarele caracteristici esențiale:

- Un interpretator de fișiere G-code care să recunoască instrucțiuni pentru diferite tipuri de mișcări și să le convertească în traiectorii.
- Interfață grafică care afișează mișcările capului de tăiere.
- Posibilitatea de a vizualiza traiectorii complexe pe diferite tipuri de materiale.

3.2 Analiza cerințelor funcționale

1. **Interpretarea fișierelor G-code**
    - Modulul de interpretare a fișierelor trebuie să poată citi și prelucra instrucțiunile în format G-code. Aceasta include identificarea tipului de mișcare (lineară, circulară), viteza și direcția mișcării, precum și coordonatele corespunzătoare.
2. **Generarea traiectoriilor de tăiere**
    - Generarea traiectoriilor de tăiere precise pe axele X și Y bazate pe instrucțiunile G-code este crucială pentru a simula corect mișcările capului de tăiere. Algoritmii de calcul trebuie să transforme datele primite în comenzi de mișcare.
3. **Simularea grafică în timp real**
    - Implementarea unui mediu grafic care să prezinte în timp real mișcările capului de tăiere și interacțiunea acestuia cu materialul.

4. **Controlul vitezei și poziției**

    - Modulul trebuie să controleze viteza și poziția capului de tăiere, respectând instrucțiunile din G-code.

4. Design

4.1 Limbaj de programare

Python, un limbaj de programare versatil și ușor de învățat, va fi utilizat în acest proiect datorită popularității și capacității sale de a dezvolta aplicații grafice și interactive. Python este foarte eficient în crearea de interfețe grafice și aplicații de tip desktop datorită bibliotecii Tkinter.

Pentru această simulare CNC, Python va fi folosit pentru dezvoltarea logicii de simulare și a interfeței grafice. Fiind un limbaj interpretat, Python permite o dezvoltare rapidă și eficientă, facilitând testarea și ajustările codului. Tkinter, biblioteca standard pentru interfețe grafice în Python, va fi utilizată pentru a construi interfata grafica a aplicației.

De asemenea, Python va fi utilizat pentru logica de procesare a fișierelor G-code. Limbajul Python este foarte eficient în manipularea fișierelor text, iar prin utilizarea unor librării va analiza fișierele G-code, extrăgând comenzile și parametrii relevanți, cum ar fi coordonatele și viteza de deplasare a capului CNC.

4.2 Structura programului

Programul este impartit in 3 module principale care se ocupa de functionarea simularii si simularea fisierului G Code.

1. **Modulul de interpretare G-code**
    - In intermediul acestei clase se vor interpreta comenzile, extrăgând coordonatele și parametrii (feed rate, poziția pe axele X și Y, unghiuri de rotație). In functie de tipul instructiunii realizate, se afiseaza un mesaj reprezentativ in consola. Toate functiile din aceasta clasa au ca scop comun interpretarea diferitelor comenzi.
2. **Modulul de Simulare Grafică**
    - Este responsabil pentru crearea interfeței grafice care vizualizează procesul de tăiere in urma parcurgerii codului. Acesta face legatura cu celalalte doua module pentru a reprezenta grafic un material (metal) fiind taiat prin comenzile incarcate prin fisier. Aici se ofera posibilitatea de a incarca fisiere G-code si de a ne deplasa pe suprafata materialului in cazul in care coordonatele de taiere se afla in afara ferestrei.
3. **Modulul de control**
    - Modulul de control se ocupa de parsarea fiecarei linii din codul incarcat pentru a se trimite linie cu linie in clasa responsabila de executia G-code. Aici se creeaza si legatura intre tipul comenzii si numarul asociat in limbajul G-code.

4.3 Interfata de vizualizare

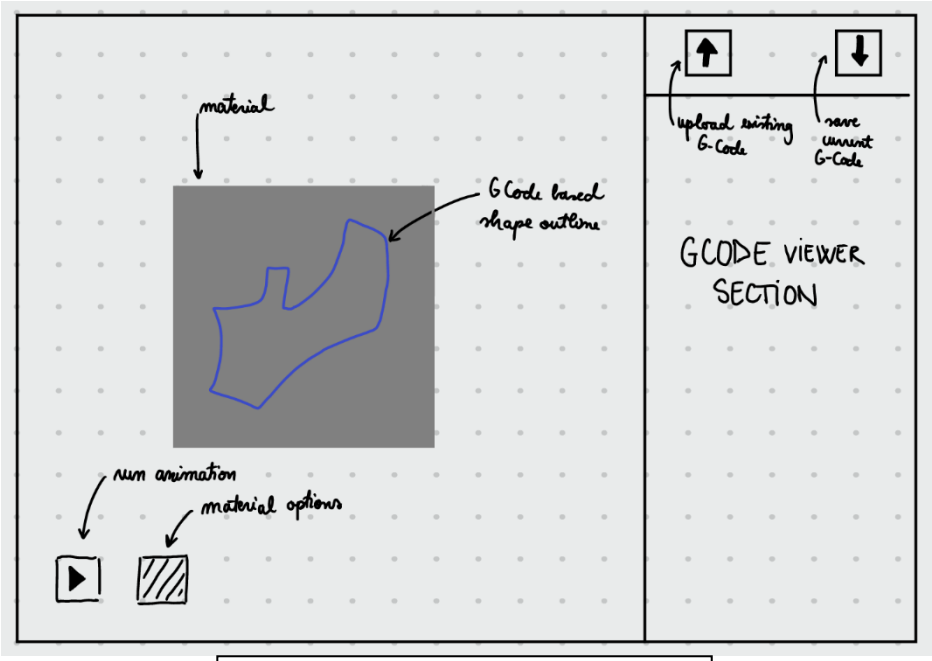
Inițial, proiectul prevedea un vizualizator de G-code care să permită modificarea acestuia în timp real. Totuși, pentru a evita posibile erori legate de parsarea incorectă a codului sau neidentificarea comentariilor, am optat pentru o abordare simplificată, în care utilizatorul încarcă G-code dintr-un fișier separat, printr-un buton dedicat.

În locul unei zone delimitate pentru materialul tăiat, cum era propus inițial, am ales să folosesc întregul spațiu de vizualizare pentru a reprezenta materialul tăiat. Astfel, accentul se pune pe modul în care G-code-ul este interpretat și aplicat, fără a limita spațiul vizual la o zonă specifică.

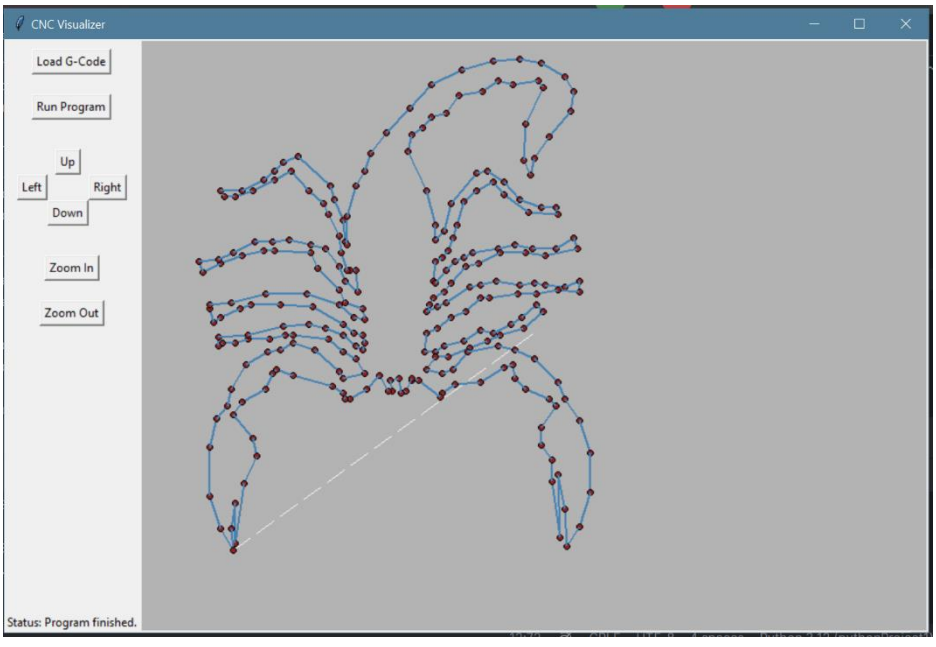
Pentru a facilita navigarea și vizualizarea detaliată a proiectului, am introdus butoane pentru deplasare și zoom in/out. Acestea permit utilizatorului să vizualizeze clar orice complexitate a rezultatului, indiferent de dimensiunea acestuia. În plus, am adăugat delimitatoare sub forma unor cercuri pe suprafața proiecției, care marchează începutul și sfârșitul fiecărei traiectorii, oferind o urmărire mai ușoară a mișcărilor.

Un alt element important al designului este reprezentarea mișcării capului de tăiere, care este afișată printr-o linie punctată albă. Această linie indică traiectoria pe care o va urma capul de tăiere, fără a interacționa efectiv cu materialul, oferind utilizatorului o imagine clară a procesului de tăiere.

De asemenea, am implementat mesaje pop-up care informează utilizatorul în cazul unor erori de încărcare a codului, cum ar fi lipsa delimitatoarelor, formatul neacceptat sau imposibilitatea de a parsa codul. Aceste mesaje oferă informații suplimentare despre numărul de comenzi încărcate, facilitând astfel un flux de lucru mai transparent și mai ușor de gestionat.



Design initial



Design final

5. Implementare

5.1 Introducere

Implementarea acestui proiect CNC vizualizează și simulează mișcările unei mașini CNC pe baza unui fișier G-code. G-code-ul este un limbaj de programare folosit pentru a controla mașinile CNC, iar scopul acestui proiect este de a încărca, analiza și vizualiza mișcările într-o interfață grafică. Proiectul este realizat folosind Python, Tkinter pentru interfața grafică și o clasă MachineClient pentru simularea comportamentului mașinii CNC.

5.2 Structura Codului

Model-View-Controller (MVC) este un pattern arhitectural software care separă aplicațiile în trei componente distincte: **Model**, **View** și **Controller**. **Modelul** reprezintă logica aplicației și datele acesteia, fiind responsabil de manipularea și stocarea datelor. **View-ul** se ocupă cu afișarea datelor către utilizator și gestionarea interfeței vizuale. **Controller-ul** intermediază interacțiunile dintre Model și View, gestionând inputul utilizatorului și actualizând modelul sau vizualizarea în funcție de acțiunile acestuia. Această separare ajută la o mai bună organizare a codului, facilitând întreținerea, testarea și extinderea aplicațiilor.

Proiectul este organizat într-o structură modulară și clar definită, compusă din trei clase principale care sunt interdependente, dar fiecare îndeplinește un rol specific. Structura este inspirată din principiile arhitecturii Model-View-Controller (MVC). În cazul nostru, aceste principii sunt aplicate într-un mod simplificat, iar fiecare componentă are roluri clare care contribuie la funcționarea generală a simulării CNC. Cele trei module principale ale aplicației sunt: MachineClient, CNCVisualizer, și Main.

5.2.1 MachineClient

Clasa **MachineClient** joacă un rol esențial în simularea comportamentului unei mașini CNC în cadrul acestui proiect. Ea face parte din partea de **Model** a arhitecturii **MVC**, deoarece gestionează logica aplicației, adică parametrii și acțiunile fundamentale ale unei mașini CNC, cum ar fi poziția capului de taiere, viteza de alimentare și controlul turației (spindle). Clasa oferă o serie de metode care permit manipularea acestora și sunt folosite ulterior în Controller pentru a răspunde la comenzile primite din fișierele G-code.

Exemple de cod relevante:

- Modificarea planului de referinta pentru proiectarea relevanta. Fiind o aplicatie care poate reprezenta doar 2D, aceste metode pot fi utilizate pentru a reformata pozitia din care ne referim la rezultatul taierii.

```
def set_plane_xy(self, params={}):  2 usages (1 dynamic)
    self._plane = PLANE_XY
    self.statusprint("Plane set to {}".format(NAMES[self._plane]))

def set_plane_zx(self, params={}):  1 usage (1 dynamic)
    self._plane = PLANE_ZX
    self.statusprint("Plane set to {}".format(NAMES[self._plane]))

def set_plane_yz(self, params={}):  1 usage (1 dynamic)
    self._plane = PLANE_YZ
    self.statusprint("Plane set to {}".format(NAMES[self._plane]))
```

- Deplasarea pe diferite axe si modificarea pozitiei acelei coordonate.

```
def move_x(self, value):  5 usages
    self.statusprint("Moving X to {:.3f} [{}]."
                     .format( *args: value, NAMES[self._unit]))
    self._pos["x"] = value

def move_y(self, value):  5 usages
    self.statusprint("Moving Y to {:.3f} [{}]."
                     .format( *args: value, NAMES[self._unit]))
    self._pos["y"] = value

def move_z(self, value):  5 usages
    self.statusprint("Moving Z to {:.3f} [{}]."
                     .format( *args: value, NAMES[self._unit]))
    self._pos["z"] = value
```

- Exista si metode care in interfata grafica nu ar avea un efect vizibil, precum schimbarea uneltei de taiere, utilizarea lichidului de racire, utilizarea starii masinii intr-un anumit punct, etc. Pentru acest tip de instructiuni am utilizat doar o metoda „statusprint” care afiseaza in consola starea masinii la acel timp.

```
def change_tool(self, tool_name):  1 usage (1 dynamic)
    self._tool_name = tool_name
    self.statusprint("Changing tool '{:s}'.".format(self._tool_name))

def manual_tool_change(self):  1 usage (1 dynamic)
    self.statusprint("Manual tool change to '{}' requested".format(self._tool_name))

def coolant_on(self):  2 usages (2 dynamic)
    self.statusprint("Coolant turned on.")
    _coolant_on = True
```

- Metodele cele mai relevante sunt cele de deplasare, respectiv deplasare lineara, deplasare rapida, sau deplasare directa. Acestea sunt si cele mai complexe. Spre exemplu, am atasat metoda de deplasare lineara, fiind si cea mai utilizata in general in G-code.

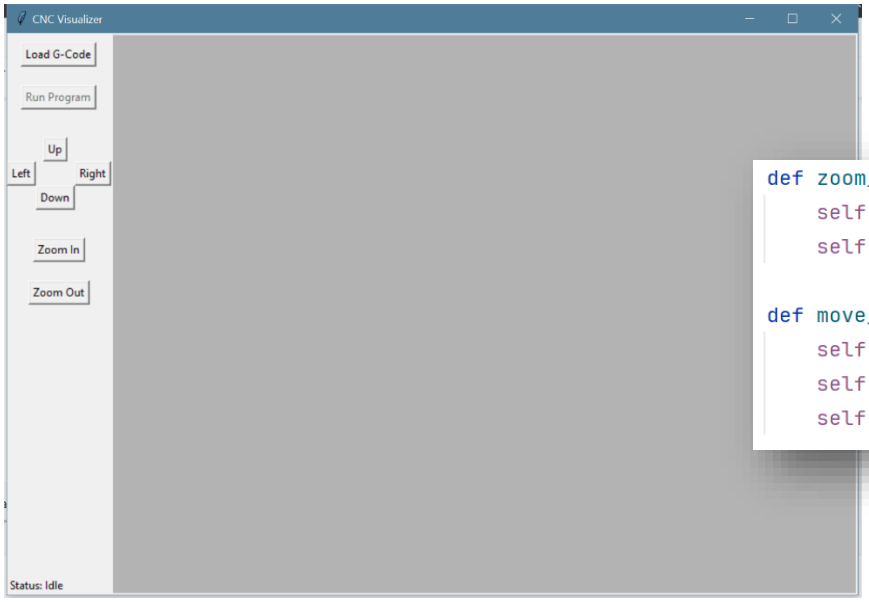
```
def lin_move(self, params):  2 usages (1 dynamic)
    self._motion_mode = MOTION_MODE_LINEAR
    if (params is None):
        self.statusprint("Setting motion mode to {}".format(NAMES[self._motion_mode]))
        return
    new_x = self._pos["x"]
    new_y = self._pos["y"]
    new_z = self._pos["z"]
    for par in params:
        if par[0] == "F":
            self.set_feed_rate(float(par[1: len(par)]))
        if par[0] == "X":
            new_x = float(par[1: len(par)])
        if par[0] == "Y":
            new_y = float(par[1: len(par)])
        if par[0] == "Z":
            new_z = float(par[1: len(par)])
    self.move(new_x, new_y, new_z)
```

5.2.2 CNCVisualizer

Clasa **CNCVisualizer** reprezintă **View-ul** în arhitectura **MVC**, având rolul de a crea și manipula interfața grafică a aplicației. Utilizând biblioteca Tkinter, aceasta creează o fereastră de aplicație cu un canvas pe care sunt desenate mișcările uneltei CNC, vizualizând astfel procesul de tăiere. Clasa include, de asemenea, controale de interacțiune cu utilizatorul, cum ar fi butoane pentru încărcarea fișierelor G-code, rularea programului și controlul zoom-ului pe canvas. Aceste elemente permit utilizatorilor să controleze și să observe în mod interactiv simularea procesului CNC.

Exemple de cod relevante:

- Partea de initializare a clasei include partea de formare a interfetei vizuale in structura atasata. Fiecare buton are o functie care raspunde la actiunea respectiva.




```
def zoom_out(self):  1 usage
    self.scale_factor /= 1.2
    self.redraw_canvas()

def move_canvas(self, dx, dy):  4 usages
    self.canvas_offset["x"] += dx
    self.canvas_offset["y"] += dy
    self.redraw_canvas()
```

- Scalarea codului pentru a putea fi vizualizat in interfata.

```
def handle_offset(self, params):  1 usage
    for param in params:
        axis = param[0]
        value = float(param[1:])
        if axis == "X":
            self.current_pos["x"] = (value - self.bounds["x_min"]) * self.scale_factor + self.canvas_offset["x"]
        elif axis == "Y":
            self.current_pos["y"] = (value - self.bounds["y_min"]) * self.scale_factor + self.canvas_offset["y"]
        elif axis == "Z":
            self.current_pos["z"] = value
```

- Incarcarea fisierului si transmiterea acestuia spre main pentru a putea fi procesat

```
def load_file(self):  1 usage
    filepath = filedialog.askopenfilename(filetypes=[("G-Code Files", "*.txt *.nc *.gcode")])
    if not filepath:
        return
    try:
        with open(filepath) as f:
            if not check_markers(f):
                raise ValueError("Invalid G-code file markers.")
            parse_file(f, self.pgm_data)

        self.calculate_bounds()
        messagebox.showinfo( title: "Success",  message: f"Loaded {len(self.pgm_data['commands'])} command blocks.")
        self.run_button.config(state=tk.NORMAL)
    except Exception as e:
        messagebox.showerror( title: "Error",  message: f"Failed to load G-code file: {e}")
```

- Accentul in partea de vizualizare este sa fie reprezentate cu acuratete deplasarile pe material (G00 si G01). In acest caz, interfata comunica direct cu machine client.

```
def execute_visual_command(self, cmd_data):  1 usage
    cmd = cmd_data["cmd"]
    params = cmd_data.get("params", [])

    if cmd in ["G00", "G01"]:  # Movement commands
        self.handle_movement(cmd, params)
    elif cmd == "G92":  # Set offset command
        self.handle_offset(params)
    execute_command(self.machine, cmd_data)
```

5.2.3 Main

Fișierul **main.py** reprezintă **Controller-ul** în arhitectura **MVC**. Acesta joacă un rol esențial în coordonarea interacțiunii dintre utilizator și restul aplicației. Responsabilitățile principale ale acestui fișier sunt gestionarea fluxului de date, încărcarea fișierelor G-code, validarea acestora și transmiterea informațiilor către **MachineClient** și **CNCVisualizer** pentru a fi procesate și vizualizate corespunzător. În esență, **main.py** pune în mișcare întreaga aplicație, asigurându-se că datele sunt corect transmise între componentele Model (MachineClient), View (CNCVisualizer) și interacțiunea utilizatorului.

- Exista un handler care mapeaza fiecare comanda la functionaltatea ei si este executat astfel. Comanda executata trebuie sa respecte prin a incepe cu una dintre cele 4 litere specifice G-code (G,M,T sau S). Fiecare comanda are un handler al sau.

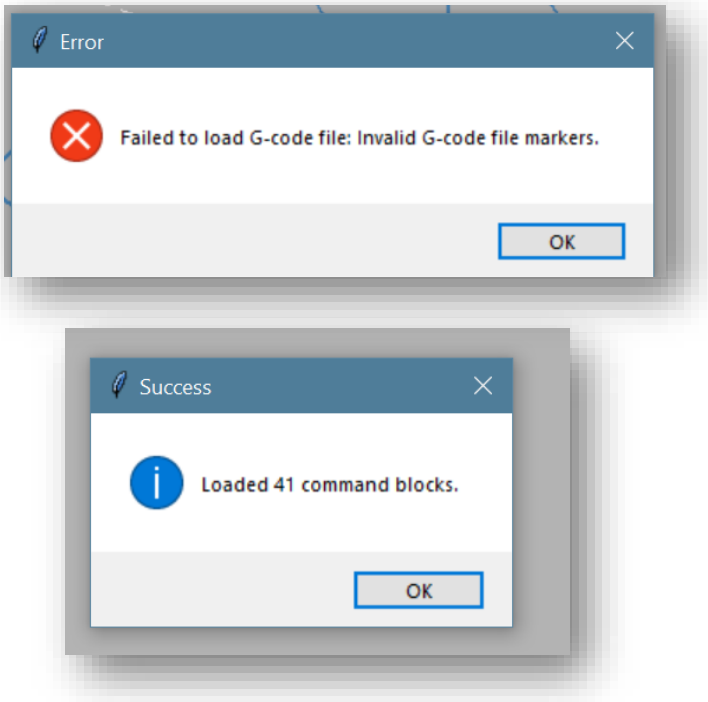
```
def execute_command(machine, cmd_data):  3 usages
    command_map = {
        "G": handle_g_command,
        "M": handle_m_command,
        "T": handle_t_command,
        "S": handle_s_command,
    }

    cmd = cmd_data["cmd"]
    cmd_num = cmd[1:]

    handler = command_map.get(cmd[0])
    if handler:
        handler(machine, cmd_num, cmd_data.get("params"))
    else:
        print(f"Unknown command {cmd}")
```

- Fiecare handler este conceput pentru a gestiona uniform comenzile G-code, mapând fiecare comandă care începe cu litera respectiva la funcțiile corespunzătoare din clasa MachineClient.

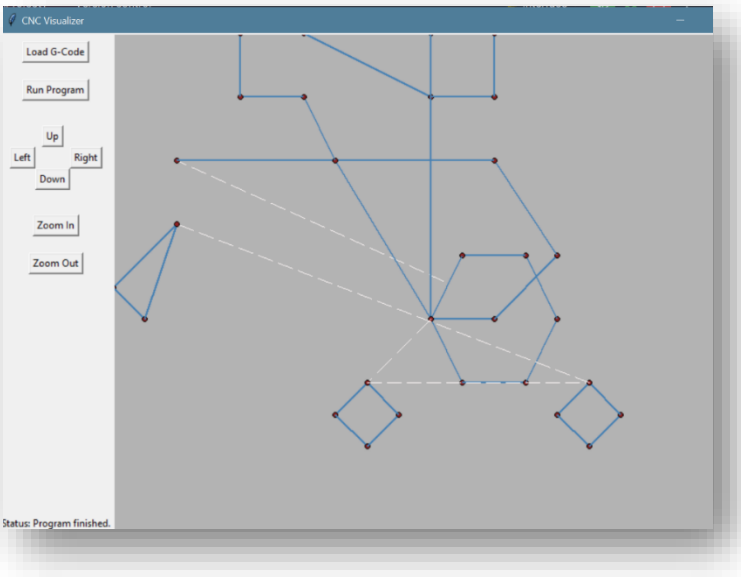
```
def handle_g_command(machine, cmd_num, params):  1 us
    G_COMMANDS = {
        "00": machine.rapid_move,
        "01": machine.lin_move,
        "17": machine.set_plane_xy,
        "18": machine.set_plane_zx,
        "19": machine.set_plane_yz,
        "20": machine.set_unit_inch,
        "21": machine.set_unit_mm,
```

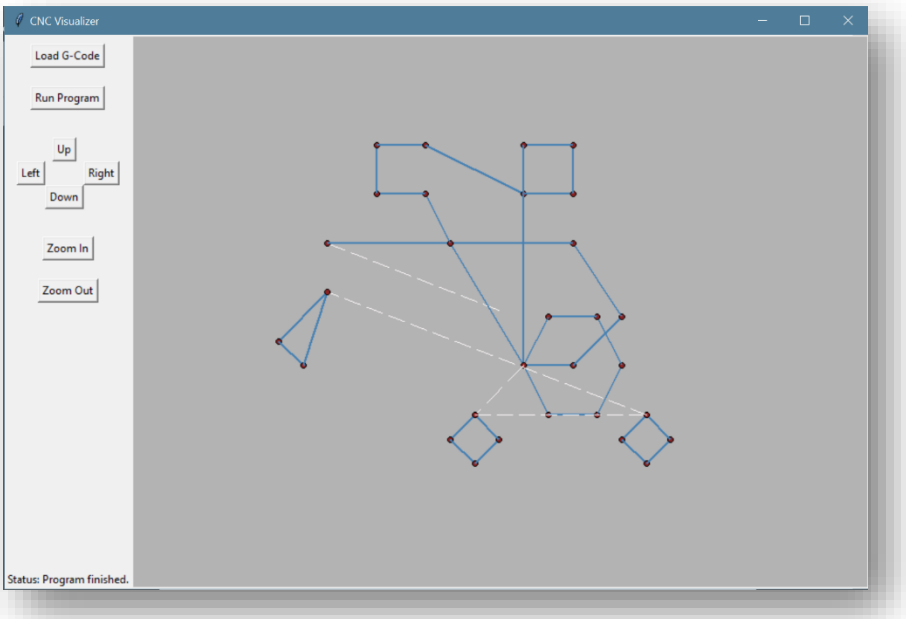
5.3 Exemple de Utilizare

Mesajele pop-up sunt utilizate pentru a oferi utilizatorului feedback imediat asupra stării aplicației. De exemplu, un mesaj pop-up poate apărea atunci când fișierul G-code a fost încărcat cu succes, indicând numărul total de comenzi din fișier, pentru a confirma utilizatorului că datele au fost procesate corect. De asemenea, pot apărea mesaje de eroare, precum „Marcator G-code neidentificat”, atunci când fișierul G-code nu conține delimitatori corespunzători (%). Aceste mesaje ajută utilizatorul să înțeleagă progresul aplicației și să identifice eventualele probleme.



Funcționalitatea de zoom permite utilizatorului să mărească sau să micșoreze zona vizualizată a mișcărilor uneltei în cadrul aplicației. Prin butoanele „Zoom In” și „Zoom Out”, utilizatorul poate ajusta scala vizualizării pentru a se potrivi mai bine cu dimensiunile traseelor generate de fișierul G-code. Acest lucru este util mai ales atunci când traiectoriile sunt mai mari decât zona vizibilă inițială, permițându-i utilizatorului să exploreze întreaga zonă de tăiere. Imaginile atașate ilustrează modul în care se poate utiliza zoom-ul pentru a vizualiza complet procesul de simulare, oferind o perspectivă mai detaliată asupra mișcărilor uneltei.





6 Bibliografie

[1] mantechmachinery

[2] Myplasmcnc.com

[3] Datron.com , languages used by cnc

[4] 3erp.com, g-code introduction

[5]linuxcnc.org, g-code spreadsheet